RAND INTELLIGENT TERMINAL AGENT (RITA): DESIGN PHILOSOPHY

R. H. Anderson, et al

RAND Corporation

Prepared for:

Defense Advanced Research Projects Agency

February 1976

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER R-1809-ARPA	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Rand Intelligent Terminal Agent (RITA): Design Philosophy		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. H. Anderson and J. J. Gillogly		8. CONTRACT OR GRANT NUMBER(s) DAHC15-73-C-0181
9. PERFORMING ORGANIZATION NAME AND ADDRESS The Rand Corporation 1700 Main Street Santa Monica, Ca. 90406		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Department of Defense Arlington, Va. 22209		12. REPORT DATE February 1976
		13. NUMBER OF PAGES 62
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No restrictions

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Processing Terminals
Computer Programs
Man Machine Systems
Input Output Devices
RITA

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

see reverse side

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

A description of the design constraints, design requirements, and overall design philosophy guid' 'g the implementation of the Rand Intelligent Terminal Agent (RITA). PITA is a set of computer programs residing in a PDP-11/45 minicomputer. These programs are capable of acting as a "user agent" which can perform a variety of tasks, under either direct user control or semiautonomous operation over extended periods of time. The RITA system is designed to be widely applicable as a standalone computing resource for local text manipulation, as a limited heuristic modeling tool, and as a front end to remote computing systems and networks. Operational features in the current (January 1976) version of RITA are described. References. (JDD)

UNCLASSIFIED

ARPA ORDER NO.: 189-1
6P10 Distributed Information Systems

R-1809-ARPA
February 1976

Rand Intelligent Terminal Agent (RITA): Design Philosophy

R. H. Anderson and J. J. Gillogly

A Report prepared for

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

The research described in this Report was sponsored by the Defense Advanced Research Projects Agency under contract No. DAHC15-73-C-0181. Reports of The Rand Corporation do not necessarily reflect the opinions or policies of the sponsors of Rand research.

iia

PREFACE

Members of Rand's Information Sciences research program are currently implementing a set of computer programs called the Rand Intelligent Terminal Agent, or RITA. This research effort is part of a larger research program on advanced intelligent terminals which is being funded and coordinated by the Information Processing Techniques Office (IPTO) of the Defense Advanced Research Projects Agency (ARPA).

The present report is one of a series documenting the design and implementation of the RITA system. It presents the overall design philosophy guiding the work in progress, and is intended for a broad audience. The report is neither highly technical nor dependent on other reports in the series.

The RITA system is designed to be widely applicable as a standalone computing resource for local text manipulation, as a limited heuristic modeling tool, and as a front end to remote computing systems and networks. As such, it should be useful to persons involved with the design of interfaces to computer networks for logistics, maintenance scheduling and control, command and control systems, intelligence collection and dissemination, and remote accessing of large data bases. It should also be relevant for designers of software systems supporting administrative and command functions.

SUMMARY

The Rand Intelligent Terminal Agent (RITA) is a set of computer programs residing in a PDP-11/45 minicomputer. These programs are capable of acting as a "user agent" which can perform a variety of tasks, both under direct user control and operating semiautonomously over extended periods of time. Among these tasks are (1) filing, retrieving, and editing of data on local storage files, (2) handling interactive dialogs with external information systems available over either telephone lines or the ARPANET, (3) providing local tutorial functions and error-checking of input data, and (4) heuristic modeling of a limited subjective set of relationships.

For a user agent to be helpful to persons who are not programmers or "computer sophisticates," it must in our opinion (1) be capable of explaining its behavior upon request, (2) be capable of having its behavior modified by the user himself, (3) be able to be governed by sets of heuristics stated as rules, rather than as formal algorithms, and (4) retain a memory of tasks assigned, progress, schedules, and deadlines in spite of either scheduled or unscheduled system maintenance periods or "crashes" (unplanned system failures requiring execution of a restart procedure). The RITA system meets these design objectives primarily by using production systems: sets of predicate-action rules operating upon a data base under the guidance of a rule interpreter, or monitor.

RITA's predicate-action rules are stated in an English-like language having a restricted set of options. The data base consists of a set of objects, each having a set of named attributes. Each attribute may in turn have either a scalar value or set of values associated with it. The form of the rules and the data base are deliberately simple to allow understanding of their structure by "computer-naive" users. Each production rule may be thought of as a heuristic guiding the behavior of a user agent. The RITA monitor is capable of either a pattern-directed scan, in which the predicates of rules are tested for applicability, or a goal-directed mode, in which the action parts of rules are scanned for relevance in achieving a specified goal.

The RITA system design consists of (1) a kernel containing the monitor, data base, and rules, and (2) a front-end module with text-manipulation capabilities for the creation and editing of rules, and a set of commands for the initiation, interruption, and querying of user agents. A basic form of RITA is currently operational, but work is still under way on several design features discussed in this report. Several examples of rule sets are given to illustrate both pattern-directed and goal-oriented modes of operation.

ACKNOWLEDGMENTS

Robert Greenberg is responsible for many key features in the design and implementation of RITA's user interface module and its syntactic analysis routines. The project has received support and encouragement from our prior ARPA program manager, Dr. Craig Fields, and our current program manager, Dr. Steve Walker; this support is greatly appreciated. Very useful suggestions on improving the form of this report were made during a careful and thorough review by Rand staff members Dr. Gabriel Groner and Ray Pyles.

CONTENTS

GLOSSARY

Algorithm	A complete set of instructions sufficient for the accomplishment of a task. It is usually represented as a computer program or a set of statements in a formally defined language.
ARPANET	A nationwide data communication network using leased telephone lines linking over 50 computers of various types. It employs a digital "packet-switching" technique allowing shared use of communication lines.
Backtracking	A method of operation for computer programs in which failure to achieve a desired goal leads the program to "back up" and try some other alternative at an earlier point at which a choice was made. Backtracking is used by the RITA monitor when an attempt is made to find a correspondence between objects mentioned in the pattern part of rules and objects in the data base. If a particular binding of data objects to references in rules has been made by the monitor, and some later predicate is found to be false for that particular binding, the system then backs up by undoing that binding and trying another binding until either a successful one has been found or all possible bindings have been tried.
Backward-chaining	A mode of monitor operation in a production system in which the next rules to be applied are chosen because their action parts set attribute values tested by a (sub) goal rule. Thus a chain of relevant rules is created "backward" from a designated goal rule. This mode of operation is an alternative to a pattern-directed mode.
BNF	"Backus-Naur Form" -- a formalism for describing grammar rules. A set of such grammar rules defines the syntax for a formal language. (A "formal language" is contrasted with a "natural language" such as English, for which no known finite set of grammar rules is sufficient to describe completely the syntax felt to be "correct" by a native-born speaker.)
Conflict set	The set of production rules each of whose pattern part (LHS) evaluates to "true" at a particular time. One or more of these rules must be selected for the subsequent execution.
Context	The data base which is part of a production system. In RITA it contains an unordered set of data objects, each having named attributes with associated data values.

Context-free	A type of formal language whose syntax can be represented by a simple grammar. See Hopcroft and Ullman [1969] for a discussion of formal languages and their properties
Goal-directed	A method of operation in which the behavior of a computer program is governed by its attempts to achieve a specified goal. It usually does this by deriving a set of requisite subgoals (and sub-subgoals, etc.) until it reaches a set of primitive tasks which can be accomplished.
Heuristic	A "rule of thumb" to be used to guide behavior (of a computer system or a person), but which is not guaranteed to always produce the desired solution under all or any circumstances. One method of encoding a heuristic to guide a computer's behavior might be as a RITA production rule.
Level of certainty	A real number between -1.0 and 1.0 (inclusive) which can be associated with an attribute value to indicate the degree of certainty in the correctness of that value. The value 1.0 indicates absolute certainty; -1.0 indicates certainty that the given value is not the correct one. Intermediate values indicate confirming (+) or disconfirming (-) evidence has been amassed for a particular value.
LHS	Left-Hand Side of a production rule, consisting of predicates to be evaluated.
LISP	A computer language and software system specializing in the storage and manipulation of data consisting of lists of items. Each item may be either a primitive value or itself a list of items.
LR(1)	An acronym for "Left to Right scan with 1-symbol look-ahead." This is a property of a grammar defining the syntax of a formal language. Sentences in a LR(1) language can be parsed, or interpreted, in one left-to-right scan of the symbols comprising that sentence by making decisions at each symbol encountered based only on the sequence of symbols previously encountered and in addition by "looking ahead" only one symbol to the right -- i.e., to the next symbol to be read.
Microprocessor	A computer consisting of one or several "chips" containing large-scale integrated (LSI) circuits or memory. The computer would thus probably be extremely small (e.g., several cubic inches, excluding such components as the power supply).
Minicomputer	A computer of modest size and cost, usually ranging in price between $2000 and $100,000.

Monitor	A computer program which evaluates a set of production rules by testing predicates within those rules and executing the corresponding actions specified in rules that are found to be applicable.
MYCIN	A computer program developed by E. H. Shortliffe and associates at Stanford University that advises physicians regarding antimicrobial therapy through an interactive dialog. It uses production rules and a goal-directed backward-chaining mode of operation; it is written in LISP. See Shortliffe et al. [1973, 1974a, 1974b, 1975a, and 1975b].
New York Times Information Bank	An information retrieval service provided by a subsidiary of the New York Times Corporation. Abstracts of articles appearing in the *New York Times* and selected other publications are available in digital form via telephone lines through keyword-based search requests.
Object	In the RITA system, a datum consisting of a name (or type), and zero or more named attributes. All attributes associated with an object must have mutually distinct names. Each attribute has an associated value, which is either a character string or a set of values.
Pattern-directed	A method of operation for production systems in which the operation of the system is governed by testing of the pattern part, or left-hand side, of the production rules. Rules whose LHS evaluate as true have their corresponding action part, or RHS, executed.
PDP 11/45	A minicomputer produced by the Digital Equipment Corporation, with an add time of 300 nanosec and maximum addressable storage of 131,072 16-bit words.
Production rule	A statement of the form: If: predicate_1 & predicate_2 & ... predicate_m Then: action_1 & action_2 & ... & action_n Depending on the particular strategies incorporated in the rule interpreter, or monitor, the actions are (at least potentially) performed when all the predicates are true. The predicates test various objects and their attribute values within a data base, or context.
Production system	A set of production rules, a rule interpreter (or monitor), and a data base (or context).

Protocol	The set of commands and responses which constitute allowable forms of communication between an information system and an external user (which may be either a person or another information system).
RHS	Right-Hand Side of a production rule, consisting of actions to be performed.
RITA	Acronym for Rand Intelligent Terminal Agent, a set of computer programs residing in a PDP 11/45 minicomputer capable of acting as a "user agent" to perform such tasks as handling interactions with remote information systems and local text retrieval and storage.
Syntax-directed parser	A computer program which receives as input the formal grammar for a language (usually in the form of a set of rules), and interprets strings of characters according to the rules in that grammar.
UNIX	An operating system for the PDP-11 minicomputer developed at Bell Laboratories. See Ritchie and Thompson [1974].

I. INTELLIGENT TERMINALS

This report discusses a design philosophy for a set of computer programs expected to reside in an intelligent terminal. The programs comprise what we call the Rand Intelligent Terminal Agent, or RITA.

What is an "intelligent terminal agent" and why is it useful? The answer involves many aspects of the way people interact with computer systems as well as new alternatives becoming possible through advances in both hardware and software technology. The following observations concerning man/machine interaction and its supporting technologies form the basis for our research:

1. Until recently, most users of information systems have been either "computer sophisticates"--such as computer programmers--or else users of systems, such as airline reservation systems, with a very limited set of options. However, with the continuing rapid decline in the cost of computer hardware and data communications, many interactive computer-based information systems are becoming cost effective for much broader categories of users. These newer systems will greatly expand the number of people interacting with computers in their daily activities, and will give access to a complex variety of interactive protocols, interfaces, command languages, and remote computing systems. People are going to need assistance in tailoring this variety of options to their specific needs and especially in freeing them from routine interactions and protocols which are not directly relevant to the content of their task.

2. Projected computer hardware cost trends and advances in microprocessor technology make it extremely likely that interactive computer terminals can be produced within five to seven years containing equivalent processing power to a present-day minicomputer, at a cost which is reasonable, assuming fairly intensive use of dedicated terminals by professionals as part of their job.

3. It is important to have certain information storage and handling capabilities locally within the terminal itself:

- Text editing and auxiliary functions such as the retrieval and storage of textual information which is currently in use. This service should be provided locally for the following reasons:

  High speed, reliable response. A local processor can give "instantaneous" feedback to simple commands (e.g., the "display next line" button) and such user actions as stylus pointing or dragging. A remote time-shared computer cannot always give such immediate feedback, and the response time tends to be somewhat erratic.

  Lower cost. Through stand-alone operation for many text-manipulation operations, use of external computers and communication systems can be minimized. Also, routine processing requiring the use of remote systems (e.g., for archival storage and retrieval of documents) can be initiated by an intelligent terminal during off-peak hours, when lower rates are usually in effect. We expect that within about five years these savings will more than offset the cost of an intelligent terminal, at least under conditions of intensive use.

  Reliability. There are fewer serial components, each of which must be in operation, for the system to be functional. (For example, use of a remote text editor on ARPANET typically requires a local host, local Interface Message Processor (IMP), communication path, remote IMP(s), remote host--all simultaneously operational.)

  Security. A terminal with local processing power can perform many needed data-manipulation operations in a "locked office" stand-alone mode, with no information transmission susceptible to compromise.

- Handling tutorial functions, answering queries for help, providing exercises, and giving local, immediate-feedback error-checking on input commands and data. Immediacy of response (e.g., to simple input error conditions) is our primary reason for advocating that these functions be provided locally.

If local computing power is available within a terminal, several other services can be provided at a small incremental cost. Such services could include:

- o Aid in interfacing with external information systems, such as the ARPANET or New York Times Information Bank, where much of the interactive protocol involves supplying standard responses which are not directly relevant to the task being accomplished. It should be possible to instruct an intelligent terminal how to handle such interactive protocols automatically, including instructions on dealing with certain error conditions, so that these details need not be remembered and handled manually by the user.
- o The ability to define and set in motion "user agents." Such an agent could:

  Look at a calendar of events and start up services for the user automatically at certain times and dates. By manipulating calendar items, the human manager can progressively modify the plan being executed by the machine. For example, by changing the due date of a report, the schedule will be automatically altered for reminders and follow-up queries to persons making contributions.

  It can monitor the occurrence of various types of events, such as the arrival of a certain piece of network "mail," or the occurrence of a certain datum in a changing data base.

  It can deliver "interactive letters" to other users' terminals; these letters are capable of carrying on a dialog with the recipient, while in the process extracting information from him in a standard format suitable for further automated processing. Figure 1 illustrates the possible operation of an interactive letter.*

---

*The idea of an interactive letter and the type of dialog shown in Fig. 1 are taken from an unpublished manuscript by Thomas A. Standish (U.C. Irvine), entitled "Scenarios for Use of an Intelligent Terminal," August 12, 1974.

```
     Jones finds a message on his intelligent terminal
indicating that an interactive letter has been received
from Jane Smith.  Jones activates the terminal and it
starts to type:

     Dear Bill,
          It is time to make plans for next year's project
   budget, and Jack asked me to coordinate everything this
   year.  It would be helpful if you could answer a few
   questions:

   How many trips to the East Coast do you expect will be
   required by you and members of your staff?  Number=
                                               ------

Here the letter stops for Jones to indicate his answer.
He places a quick telephone call to a key subordinate to
verify his plans, then adds other trips he knows will be
necessary, and types in "6".  The letter continues:

   Please give the names of consultants you expe   to use
   during the next year, and the number of days' support
   required for each of them.  (When the list is finished,
   respond to "Name" with a carriage return.)  Name=
                                               ----

Bill looks at his plans for the forthcoming year, calls a
consultant whose availability was uncertain at their last
discussion, then types "A.C. Johnson".  The letter responds:

   #days=
   -----

Bill types "20", then repeats the cycle for several more
consultants, finally terminating with a carriage return as
a response.  The letter then continues:

   What is your estimate of your requirements for computer
   services during the forthcoming year?  Please give a
   dollar amount.  Answer=
                   ------

Bill types "$48,000." and the letter concludes:

   Thank you for your help.  I hope to have a draft project
   budget in your hands for review by next Wednesday.

                             Regards,
                             Jane Smith
```

Fig. 1--Illustration of the operation
of an interactive letter

It could be responsible for managing transactions between a number of computerized services distributed on a computer network, monitoring their successful accomplishment.

We believe that the desirability of the above list of services is a compelling reason to explore the design of intelligent terminal agents capable of running in present-day minicomputers. Our work on the RITA system is aimed at developing a prototype agent system capable of performing all of the above activities.

What specific system design requirements are implied by the above discussion? The next two sections of this report list several design constraints within which this research is being conducted. Then, in that context, we itemize a set of design requirements for an intelligent terminal agent which we have distilled from the general characteristics described above; these requirements are the basis for our design decisions during the implementation of the RITA system.

II. DESIGN CONSTRAINTS

There are a number of constraints on our design and development of an intelligent terminal agent which are a natural consequence of such factors as our source of funding and available computational resources. Some of the most important of these are listed below.

1. This research is being funded by ARPA-IPTO because of its prospective benefits to the Department of Defense (DOD). A particular user group within DOD targeted by ARPA-IPTO as an initial testbed for intelligent terminal systems is analysts within the intelligence community. We therefore view our initial RITA system as an intelligence analyst's station, and characteristics of this user group (e.g., well-educated and requiring text-manipulation tools and access to a variety of external information systems) have influenced our design decisions.

2. Because our concept of an intelligent terminal agent presupposes minicomputer-like power becoming locally available to a user, the RITA system must be capable of running in a present-day minicomputer. An additional reason for developing RITA in an existing minicomputer is that the software might be capable of transfer directly into a terminal of the future, if that terminal's built-in computer copies, or emulates, the instruction set and architecture of RITA's present host machine.

3. There is a PDP 11/45 minicomputer at Rand for computer research. It runs the UNIX operating system [Ritchie 1974] (developed at Bell Laboratories) which supplies many relevant facilities, such as time-sharing, interprocess communication, a system programming language (called "C") with many advanced facilities, a hierarchical file system, and so forth. The PDP 11/45 with UNIX is the obvious choice for the initial implementation of RITA.

4. An interactive man/machine interface called the Rand Editor is under concurrent development at Rand on the PDP 11/UNIX system. This editor provides excellent two-dimensional cursor-controlled text-manipulation, editing, and storage facilities. We expect to integrate the Rand Editor with the production system described in this report to

produce a more complete RITA system which will allow use of the text editing facilities for the creation and modification of production rules. The "text window" concept embodied within the Rand Editor should provide multiple windows for communication with one or more user agents, possibly running concurrently under the control of production system monitors. (A recent doctoral thesis by Swinehart [1974] proposes a similar but more elaborate form of interactive user interface with multiple concurrent processes.)

III. DESIGN REQUIREMENTS FOR AN INTELLIGENT TERMINAL AGENT

We have extracted the following specific design requirements from the discussion of general features in Sec. I, taking into consideration the constraints listed in Sec. II.

1. Most users of such terminal agents will be computer-naive. They will be given a terminal system which has been tailored to the perceived needs of a class of users by application specialists who are expected also to have programming skills. This basic system must be capable of performing actions for the user and of explaining its behavior, upon request. (We feel that a computer-naive user will not trust an intelligent terminal agent to perform complex tasks, such as logging into remote computer systems for him to transfer data out of his "mail" files, without his being able to ask the agent what actions were taken, and why.)

2. There must be a language through which the behavior of the agent can be modified and extended by a user. Traditional programming languages do not seem appropriate for this purpose because:

- o The user will probably not be a programmer.
- o The user is not expected to think in terms of algorithms as a means of instructing his terminal agent, but rather in terms of sets of rules, or heuristics, in accordance with which it should operate. (These heuristics will be supplied explicitly by the user, at least in initial versions of the system; in later versions, the system might be capable of acquiring new rules by example or through induction.)
- o The nested control structures of programming languages are unnatural and a source of error to computer-naive persons (see Miller 1975 for a further discussion).

3. The agent must be capable of two-way communication both with the user and with external systems (e.g., over dial-up telephone lines or an ARPANET connection).

4. It must be capable of running in a minicomputer.

5. The system must be capable of retaining a "memory" of tasks assigned, schedules, deadlines, and so forth in spite of scheduled maintenance periods or unscheduled system crashes.

6. The system should allow the retrieval, editing, and storage of text, and have an understanding of calendar and clock time to form the basis for handling appointments, scheduling, and other time-management tools.

IV. PRODUCTION SYSTEMS AS THE BASIS FOR A TERMINAL AGENT

The basis for our design of a system meeting the above requirements is the use of production systems. A production system consists of a set of production rules, which operate upon a data base (which we call a *context*), according to the actions of a rule interpreter, or *monitor*.

For example, two production rules might be:

Rule 1

IF: there is a message whose status is "awaiting action" and the identification-field of the message is not in the set of action-items of the user

THEN: put the identification-field of the message in the set of action-items of the user;

Rule 2

IF: The latest-command of the user is "show action items" and the state of the system is "command unfulfilled"

THEN: send the set of action-items of the user to the user and set the state of the system to "command fulfilled";

These rules would be part of a larger set of rules governing a a message-handling user agent. They might be interpreted by a monitor that continually tests the "if" conditions in each rule of the set, and executes the "then" actions in any rule whose conditions are all true. Assuming messages with various attribute values, such as an identification field and status, are placed in the data base by some external (and possibly asynchronous) process, the above rules would update a set consisting of the identification numbers of all messages awaiting action, and show that set to the user upon his request. Other rules would themselves, or permit the user to, take other actions and as a consequence change the status of the messages and remove their identification numbers from the set of action items.

In the remainder of this section, we discuss some of the options available in designing a particular production system, itemize the advantages we see accruing from their use, and then discuss the particular design decisions we have made in the choice of rule format, monitor, and format of a context data base for the RITA system. The appendix to this report contains some examples of RITA rule sets and traces of their operation. Our discussion generally follows that given in the recent excellent survey article on production systems by Davis and King [1975].

DESIGN OPTIONS IN THE USE OF PRODUCTION SYSTEMS

There is considerable variety in existing production system applications. The options available can be discussed under the headings of their three main components: context data base, rules, and monitor.

Context Data Base

The simplest form of a context on which rules may operate is a string of symbols. The rule tests for the existence of a substring within the context, and supplies a replacement string to be substituted for it if found. At the other extreme, the data base may be a complex semantic network or other form of structured data base within which rules test for patterns and upon which rule actions perform operations. The former type (i.e., string substitution) is often used by cognitive psychologists to model low-level cognitive information manipulation processes. Production systems which operate on complex data structures tend to have complex rules which are difficult for either humans or other computer programs to decipher. This would be counter to the spirit of the production system approach. Therefore, such systems must be designed with great care.

An intermediate form of data complexity which has been found to be useful, e.g., in the MYCIN system developed by E. H. Shortliffe and associates at Stanford University [1973, 1974a, 1974b, 1975a, 1975b], is a context consisting of a set of *objects*, each of which has an associated set of named *attributes*, with each attribute having an associated *value*, or set of values. This form of data has been used in innumerable LISP programs and is embodied in the property list mechanism in that language. It is also a data form used in relational data bases, where each item of information is stored as a triple (attribute, object, value). In this general data form, there are many design decisions to be made which affect the resulting complexity of the data base and the complexity of the actions needed to operate upon that data. Some options are

- o Can attributes take a single value? Or a set of values? If a set, is it unordered or ordered, and what accessing options are available for testing values and replacing values within the set?
- o Is an attribute value a scalar quantity? Or can it be (a pointer to) another object? If the latter, then the attribute acts as a named relation between two objects, but without restrictions this feature can result in a data base having pointers which form an arbitrarily complex directed graph.
- o Is there an external structure imposed on the objects in the context? The MYCIN system, for example, has found it useful to place objects in a tree-structured context; this allows incompletely specified object references within rules to be bound to the "nearest" object within the tree to the location at which the rule is currently operating.
- o Can an attribute value have a probability or confidence level associated with it?
- o Do all objects have a unique indentifier associated with them? If not (for example, if there can be numerous instances in the data base of an object called "block"), then how are specific objects referenced? Once a certain object has been referenced by one rule, can that reference be passed, either explicitly or implicitly, to other rules?
- o Can objects, and attributes of existing objects, be created and deleted dynamically by the actions of rules?

Rules

A production rule consists of a left-hand side (LHS) or *pattern* part, and a right-hand side (RHS) or *action* part. The pattern part of a rule chosen by the monitor is evaluated with respect to the current context. If it's true, the corresponding action part is executed, and another rule is chosen for evaluation. If it's false, the actions are not executed, and another rule is chosen. The simplest form of rule consists of symbol strings as both LHS and RHS, as mentioned above. In this case, the context is a string, and the evaluation of the LHS is merely a check whether the LHS is a substring of the context string. If so, that substring is replaced by the corresponding RHS.

The most general form of a rule contains arbitrary predicates on the LHS, with one or more arbitrary function calls on the RHS. The predicates test for the existence of certain data, or relationships among data, within the data base. The function calls on the rule's RHS make changes to the data base (and perhaps perform other actions as "side effects," such as emitting messages to external processes).

Several other options in the form of rules are noteworthy:

- o String pattern/replacement rules, allowing variables and "don't care" symbols as part of the pattern and replacement strings. These rules are not unlike SNOBOL [Griswold 1973] statements.
- o Use of a stylized, limited language capable of interpretation by a context-free syntax-directed parser* for expressing predicates and actions. This language might allow certain options in testing and changing the context data base, but would not permit arbitrary tests and actions whose effects on the data base could not be interpreted by the monitor itself.
- o An option similar to that above, but with a natural-language front end capable of understanding rules written by a user in natural English and of translating them into a stylized set of predicates and actions. (This option is exemplified by the MYCIN system.)

Other options to be considered in the design of rules for a production system are:

- o Whether the rules are to be used in a goal-directed or pattern-directed manner. A *goal-directed* system has a designated goal, and the objective is to execute rules whose actions achieve

*These technical terms are contained in the Glossary.

that goal. A *pattern-directed* system, on the other hand, merely tests the pattern part of rules chosen according to some scheme by the monitor against the current context, and applies (one or more of) those that match. To some extent, rules must be designed to operate according to the characteristics of a particular monitor. For example, rules used in a goal-directed system should not have action clauses causing side effects. Such side effects often cannot be undone when backtracking to try an alternative path in pursuit of the goal. The various monitor options are discussed in the following subsection.

- o The degree to which the rules capture discrete pieces of knowledge. It is advantageous in production systems to have the rules as independent of each other as possible, since one of the prime motivations for expressing process descriptions in rule form is to be able to modify those descriptions easily, and possibly even automatically. If rules do not capture discrete pieces of knowledge, so that they may be added or removed easily without destroying the logic of the system, then many of the advantages are lost.
- o The readability of the rules. Are the rules meant for machine consumption only, or for human readability also? An example of the former might be:

```
          FUN_1(X) & FUN 3(Z)  =>  EXEC(G(X));
```

An example of the latter might be:

```
    IF:  the prompt character of the remote_system is "@"
    THEN:  the name of the remote system is "tenex" (P=.7);
```

- o What is stored in rule form? It is most natural to state heuristics in the form of rules, but it is also possible to store data (e.g., "if he asks for x, give him y") and control information (e.g., rules for choosing the next rule to test).
- o Are all rules potentially applicable at all times, or is their applicability limited, for example, by partitioning them (either automatically or explicitly by the user) into sets, only one of

which is applicable at any time? (Another method of limiting the applicability of rules is through ordered rule sets, in which the applicability of a rule is governed by its position within that ordered set.)

Monitor

This discussion of monitor design options closely follows that given by Davis and King [1975]. The basic control cycle performed by a monitor consists of two phases: *recognition* and *action*. The recognition phase involves selecting a single rule for execution, and can be further subdivided into *selection* and *conflict resolution*. In the selection process, one or more potentially applicable rules are chosen from the set and passed to the conflict resolution algorithm, which chooses one or more of them for execution. The options in monitor design, then, can be discussed in terms of the above categories.

Selection. Rules can be selected by a LHS scan or a RHS scan. In a LHS scan, each rule LHS is evaluated in turn. If this process stops at the first successful evaluation encountered, then conflict resolution is trivial. It is possible, however, to collect (into a set called the *conflict set*) all rules whose LHSs evaluate successfully. (It is in fact possible to have multiple occurrences of a single rule in the conflict set, if more than one set of bindings between objects and attributes mentioned in the rule and data objects occurring in the context make the rule's LHS evaluate successfully.) It is then necessary to perform conflict resolution to choose the rule(s) for execution from this set.

Selection by a RHS scan can be considered a form of goal-oriented operation. One specific item of information (an attribute of an object) is designated as a *goal*. The goal of the system operation is to execute the actions on the RHS of rules that set this attribute value. To this end, the LHSs of those rules are evaluated. If any such LHS clause refers to an item of information not yet in the context data base, obtaining that item becomes a subgoal. A RHS scan is performed to find all rules that contain in their RHS an action clause which creates that item of information. All rules meeting this criterion are placed in the conflict set. This form of RHS scan is best exemplified by the operation of the MYCIN system of E. Shortliffe et al. Interested readers are referred to the excellent description of MYCIN's operation contained in Shortliffe's recent Ph.D. thesis [1974b].

Conflict resolution. If a conflict set has been created during the rule selection process, conflict resolution is necessary to choose which rule(s) in that set should be executed. Several possible criteria for conflict resolution (suggested by Don Waterman of Rand) are:

- o Rule order. There is a complete ordering of all rules in the system, and the rule in the conflict set with the highest priority is chosen.
- o Data order. Elements of the data base are ordered, and that rule is chosen which matches elements in the data base with highest priority.
- o Generality order. The most specific rule is chosen.
- o Rule precedence. A precedence network (perhaps containing cycles) determines the hierarchy.
- o Recency order. The most recently executed rule is chosen, or the rule containing the most recently updated element of the data base.

It is not necessary that only one rule be chosen for execution from the conflict set. In MYCIN, for example, action clauses in rules contain a *certainty factor* when creating an item of information. MYCIN executes all rules in the conflict set, using the combined certainty factors to achieve a combined judgment, which is reflected by data values (with associated resultant certainty factors) in the context data base.

Action. There are few options associated with the execution of action clauses of successful rules by a monitor. Most actions set context data values for subsequent testing by other rules. In production systems it is considered poor for actions to have complicated side effects or to execute arbitrarily complex programs, although side effects are sometimes necessary to perform such actions as communication with an external process or system. One control-type action sometimes allowed is transfer of control to a different set of rules, which can be considered another "state" of the system, in which the behavior of the system in each state is governed by a rule set. The VIS system of Moran [1973] is one example of a system allowing such clustering of rules.

Before discussing the particular design decisions we have made, choosing from the large selection of options listed above, we will consider next the advantages to be gained from using a production system. The priorities we placed on certain advantages have had an important effect on our design decisions.

ADVANTAGES OF PRODUCTION SYSTEM USE

The following advantages are often cited as accruing from the use of production systems. We have listed them in what we believe is an approximate decreasing order of importance for the particular application for which RITA is designed: namely, the construction of intelligent terminal agents.

1. *Their explanatory capability.*

> Production system rules are intended to be modular chunks of knowledge and to represent primitive actions. Thus, explaining primitive acts should be as simple as stating the corresponding rule--all necessary contextual information should be included in the rule itself. Achieving such clear explanations, however, evidently strongly depends upon the extent to which the assumptions of modularity and explicit context are met. [Davis and King 1975]

Figure 2 contains an example taken from Shortliffe [1975] of the highly advanced explanatory capability of the MYCIN system. MYCIN is an excellent example of the degree of explanatory power that can be achieved through careful design and implementation.

2. *Simple control structure.* Experiments conducted by Miller [1975] on various methods of transfer of control in programming languages have shown that if-then-else nested conditionals and branch-to-label control structures are serious sources of programming error.

```
MYCIN:    In light of the site from which the culture was
          obtained, and the method of collection, do you
          feel that a significant number of ORGANISM-1
          were obtained?

USER:     Why?

MYCIN:    [i.e., Why is it important to determine if a
          significant number of ORGANISM-1 were obtained?]

          1.0] It is important to find out whether there is
          therapeutically significant disease associated with
          this occurrence of ORGANISM-1.
          It has already been established that
              1.1] the site of the culture is not one of those
                   which are normally sterile, and
              1.2] the method of collection was sterile.
          Therefore, if
              1.3] the organism has been observed in
                   significant numbers
          then there is strongly suggestive evidence (.9) that
              there is therapeutically significant disease
              associated with this occurrence of the organism.
          [ALSO:  there is strongly significant evidence (.8)
              that the organism is not a contaminant.]
          (RULE042)

The user may then query further about why this information
is important (1.0]), in which case MYCIN would give a
similar explanation based on the rule that called this one.
```

Fig. 2--Illustration of the explanatory capability of the MYCIN system

Miller also reports that a "procedure table" specification of control led to much better performance by naive programmers. Figure 3 illustrates the format of Miller's procedure table. It is very similar to (and in fact partially derived from) production systems. Due to the simple control structure of production systems, especially of the LHS scan type, we can imagine the following type of instructions being nearly sufficient to introduce a user to the operation of his terminal:

> This terminal operates according to a set of rules. Whenever it finds a rule that is true, it applies that rule. If you want to know why it is asking you for some item of information, or why it took some action, type "?" and it will show you the rules it followed in taking that action.

If, in addition, the rules themselves are in simple English so that they are directly readable by a user, then we believe he will find the operation of this device quite understandable. Although the user will of course not understand all the nuances of its operation, he is at least not bewildered at the start, and can add incrementally to his understanding with experience. The user must realize, however, during this initial introduction to the system that there are nuances and that he should not be overly complacent or trusting of system behavior.

A RHS scan backward-chaining system, although more complex in its control structure, can give rational explanations of its behavior in a manner that makes the flow of control among rules understandable. Again, we offer the use of MYCIN by computer-naive physicians as proof for this assertion.

3. *Incremental addition of knowledge*. With proper design, production systems can allow gradual, incremental addition of knowledge and heuristics in a top-down manner. If the set of rules is unordered, as is usually the case in a RHS scan system and could be the case in a LHS scan mode, then new rules can be added to the set without concern for their placement. If the mode of operation is LHS scan through an

Procedure Table			
Label	Question	Action(s)	Go To
A1	Any card in input box? No: Stop	Look at next card	
	Name on card has second letter as "Not-L" or last letter as "N"	Put card in box #3, increase Counter 1	A1
	---------->	Put card in Box #2	A1

Problem: Put a card in Box 3 if either the name on the card has the second letter not "L" or else the last letter is "N" (or both).

Count the number of cards in Box 3 using Counter 1. Put the remaining cards in Box 2.

Fig. 3--Format of Miller's decision table, with associated problem description ordered set of rules (choosing the first true one encountered), then the location of rules to be added within that ordered set becomes important. For these reasons, we believe that unordered rule sets should be used to implement at least those portions of intelligent terminal agents which are expected to be modified to any degree by the user.

An example of the operation of the MYCIN system shows the incremental addition of knowledge. When a predicate on the LHS of some rule in the MYCIN system needs an item of information, it searches for rules whose RHS assign that datum. If such rules are found, their evaluation is attempted. If there are no such rules, the system requests that datum from the user. This provides a natural opportunity for the gradual introduction of knowledge and heuristics to the system. If the user does not wish to continue supplying that datum to the system, he has the option of giving the system a rule describing how to derive that datum from other data within the data base. In conjunction with this new rule, it might be appropriate for the user to give the system other additional rules for acquiring information from external sources. In this manner, gradual evolution of the behavior of the system takes place to meet the needs of the user in his possibly unique environment.

4. *Trainability and learning*. Assume production rules are stated in a constrained syntax so that their meaning is understandable by machines, and that each rule is a "noninteracting chunk of knowledge or behavior." Then it becomes possible for a computer program to create rules in the proper format and insert them into existing sets of rules to change the behavior of a production system. It might also modify existing rules to change a system's behavior.

Waterman [1970] has discussed the creation of new rules from training information and the process of "blending" new rules into an ordered set of existing rules. He has placed the training information a system should receive (or extract) from a user into three categories:

a. Acceptability information: an acceptable decision for a particular situation.

b. Relevancy information: the situation elements relevant to making this acceptable decision.

c. Justification information: the reason the decision is being made, expressed as an evaluation of these relevant situation elements.

The relevancy and justification information is used to create the pattern part (LHS) of a new rule, and the acceptability information is used to create the action part (RHS) of the new rule. The newly created rule is called a *training rule*. Waterman gives an algorithm for deciding whether to "blend" the training rule into an existing ordered set of rules by using it to modify an existing rule or by adding it to the existing set in an appropriate location.

In a recent paper, Waterman [1974] discusses several examples of adaptive production systems, written in the PAS-II notation [Waterman 1973], in which all adaptivity is obtained by adding new rules to existing ordered rule sets, and training information is generated internally rather than requiring feedback from a user.

5. *Unified, consistent structure*. If a production system is considered as a programming language, it is one with only a single statement type: a pattern-action rule. If, as we expect, it is as easy to program and update intelligent terminal agents in production systems as it is in ordinary high-level programming languages, then for this application production systems satisfy Occam's Razor: they are the simpler form.

In addition, it is possible to program significant portions of the monitor (e.g., the conflict resolution strategy) in a production system form, so that the structure of the entire system becomes more unified; in that case, the monitor itself becomes amenable to modification, and possibly to adaptive learning.

There are also some disadvantages in the use of production systems. The two major ones are:

1. It can be difficult to code an operation in the form of a production system, particularly for goal-oriented rule sets. Considerable thought must be given to the choice of objects, attributes, and values by which a problem area is represented. (However, the problem

of choosing a good data representation is certainly not unique to production systems; the problem lies more in trying to fit all applications into this particular procrustean bed.) One must also carefully choose certain attributes of objects to represent "state variables" which encode the state of a computation or deduction. The values of these state variables are tested by various rules to trigger their potential applicability. In this manner, production systems encode explicitly that which in ordinary high-level programming languages is implicit in the nesting of control statements. For example, a traditional programming language nested control structure such as:

```
if A then
     if B then C
          else if D then E;  else;
     else G;
```

might be encoded in a production system in the following manner:

```
if A                then state_1;

if state_1 and B    then C;

if state_1 and not B then state_2;

if state_2 and D      then E;

if not A              then G;
```

Such explicitness in a production system allows the desired relative autonomy of individual rules, but at the price of requiring the programmer to create names for many intermediate states of his process.

In the RITA system, we hope to overcome this disadvantage by having system experts create initial systems and user agents having general applicability. Individual users are expected, at least initially, to make only rather minor modifications and enhancements to the basic system. Therefore, the vocabulary and overall design of a user agent will be established, providing many guidelines and examples for the individual user.

2. Production systems are often inefficient. It is quite easy to design systems in which the pattern parts of hundreds of rules are tested against the data base before a successful match is found; it is also easy in goal-oriented systems to pursue lengthy chains of reasoning which are not useful. The same deductions may be recomputed redundantly many times in separate logic paths, without awareness in the system of the duplication of effort.

Our design of the RITA system has not been significantly influenced by efficiency considerations. The simple user agents which have been constructed to date (e.g., for handling File Transfer Protocol interactions on the ARPANET) have required only 30 to 40 rules and are not inefficient. As more complex agents are constructed, we believe there are a number of monitor enhancements that can be designed to increase efficiency (e.g., through hash-coded lookup tables to aid in finding applicable rules) which can be added as the need arises. A system called PSH, currently under development within the Computer Science Department at Carnegie-Mellon University, is a testbed for a major study of methods of obtaining efficiency in production systems.

With the above advantages and disadvantages in mind, and given the options available in production system design and the design requirements derived from the particular application under discussion, we can now discuss the design decisions made to date in the implementation of the RITA system.

DESIGN DECISIONS IN THE RITA SYSTEM IMPLEMENTATION

Our design decisions in creating a production system for our particular needs are discussed under four headings: data base, rules, monitor, and system architecture.

Data Base

The data base upon which RITA rules operate is called a *context*; it consists of an unordered set of objects. Each object has a name, or type, and there can be more than one object in the context of the same type. There is neither an external structure imposed on the set of objects in the context nor a requirement that each object have a unique identifier associated with it. Each object can have one or more named attributes, and all attributes attached to an object must have names which are mutually distinct. Each attribute has an associated value, which is either a character string or a set of values.

Objects, attributes, and values may be created or deleted dynamically by the actions of rules. If an attribute being tested by a rule's LHS predicate does not exist, it is considered to be "not known." It is possible by a rule action (except within a goal-oriented monitor) to reset an attribute having a value back to the "not known" status. Goal-oriented monitors may not reset the value of any attribute; they may only set values which were previously not known. This restriction is necessary to preserve the integrity of information upon which chains of logical reasoning are based.

As an option, it is possible to attach a "level of certainty" to a scalar attribute value as it is being set. In this case, an attribute can have several different values associated with it, each with a different level of certainty. Levels of certainty attached to values are adjusted as additional positive or negative certainty factors for those values are asserted by the action of rules. Our planned use of certainty factors has been strongly influenced by their implementation in the MYCIN system. Our implementation is expected to differ in some details, but a discussion of those differences will not be presented here.

Figure 4 contains examples of object types and associated attribute names and values which might be used in a user agent within the RITA system.

The data structure we have chosen is not the most general one possible. (An obvious extension allowing much more generality would be to allow references, or pointers, to objects as attribute values. As mentioned earlier, this would give the capability for arbitrarily named relations among objects.) As with other implementation decisions, we have chosen what we consider to be the simplest format and

object type	attribute name	sample value
file	name	"foo.baz"
	directory	"jjg"
	site_id	"rand-isd"
	size	20000
	owners_name	"gillogly"
site	id	"rand-11"
	operating_system_name	"unix"
	machine_type	"pdp-11/45"
	guest_account_name	"netguest"
	guest_account_password	"netguest"
	known-user-set	("jjg", "rha", "rsg")
known_person	name	"gillogly"
	primary_site_id	"rand-isd"
	primary_directory	"jjg"
	primary_password	"whumpus"
	secondary_site_id	"cmu-10a"
	secondary_site_directory	"g250a12"
	secondary_site_password	"foo"

Fig. 4--Examples of RITA object types, attributes, and values conceptual structure which allows the description of situations and heuristics related to intelligent terminal agents. With more experience in using the RITA system, some of these decisions are almost certain to change.

Rules

RITA rules are expressed in a finite syntax (technically, parsable by an LR(1) algorithm). A complete syntax chart for RITA rules as they now stand is given in a companion document [Anderson and Gillogly, to be published]. We have chosen a syntax patterned after the general English output form generated to display MYCIN rules to a user. We believe that this syntax is simple enough to be read and written by a computer-naive user. Figure 5 contains several examples of clauses which can be used in RITA rules; more complete examples are contained in the appendix.

Such facilities as string-manipulation are provided in the RITA system by a set of primitive functions which may be called in the predicates or actions of rules.

We note that all RITA rules, including those to be interpreted by a goal oriented monitor, are expressed in the same syntax. MYCIN, on the other hand, uses a goal rule hand-coded in LISP which does not follow normal MYCIN conventions.

We have decided, for simplicity, not to implement multiple rule sets which limit the applicability of a rule to those times when its set is the "current state" of the system. From our experience to date with the system, the additional mechanisms required for explicit transfer of control among named rule sets does not seem justified by advantages in operating efficiency.

Monitors

We have found that different types of monitors are necessary for various specific tasks and situations, and that no one monitor type is sufficient for our purposes. For example, interactions with an external information system to handle routine protocols are best handled by a LHS scan monitor, acting in what might be called a "stimulus-response" mode. On the other hand, it is sometimes necessary for an

```
                 Left-hand-side predicate clauses

   IF:   the name of the system is "unix"

   IF:   the name of the system is the name of the
             desired_system

   IF:   the name of the system is not known

   IF:   there is a response whose arrival_time is less than
             the max_expected_delay of the system

                 Right-hand-side action clauses

 THEN:   set the name of the system to "net access program"

 THEN:   set the valid_id_set of the remote_site to the
             id of every site whose id is known

 THEN:   deduce the guest_account_name of the remote_site

 THEN:   create a remote_site whose id is "cmu-10a"

 THEN:   receive the next line from the system_IO_pipe
             as the value of the response

 THEN:   send "Which rule do you wish to see" to the user

 THEN:   return success
```

Fig. 5--Examples of RITA rule clauses

intelligent terminal agent to make deductions (e.g., about the most likely site on the ARPANET for a particular person to have a mailbox, given that person's attributes). Deductions are best made by a RHS scan, goal-driven monitor.

Consequently, we have implemented several different monitors:

- o LHS scan, with ordered rule set
- o LHS scan, with unordered rule set
- o RHS scan with backward-chaining (implicitly, a rule set is treated as unordered)

Nothing in our implementation precludes the development of other monitors if the need arises. The top-level monitor in a user agent will use a LHS scan with an unordered rule set; however, it is possible for the following action clause of some rule to be executed:

DEDUCE attribute OF object.

This clause triggers the operation of the RHS scan backward-chaining monitor with the goal of deducing the value of the named attribute. Only rules in the system which are designated as RHS scan rules are used during the backward-chaining process to deduce the required information. Upon completion of a deduction, control reverts to the action clause of the LHS scan rule following the DEDUCE clause, or if there are none, to the next applicable rule chosen by the LHS scan monitor. It is not possible to explicitly invoke an LHS scan monitor during a goal-directed deductive operation.

The design of a goal-oriented backward-chaining monitor involves many unique decisions not encountered in LHS-driven monitors. As mentioned earlier, our goal-directed monitor has been heavily influenced by MYCIN, but differs from that of MYCIN primarily in the following respects:

1. In pursuing a goal without specified levels of certainty in rules, our search terminates when the desired information is first

found (rather than pursuing all possible paths, as in MYCIN, before combining the certainty factors associated with all results found).

2. The binding of objects and attributes mentioned in rules to objects in the data base is governed in MYCIN by a data hierarchy. The data mentioned in a rule are bound to a datum at a level in that hierarchy dependent on the "lowest" object mentioned in the rule. From that binding, a context is inherited from the hierarchy which can resolve many possible ambiguities and search requirements. In RITA, on the other hand, our data context is not structured. Consequently, no implied context is used to resolve searches.

3. In MYCIN, an automatic search is performed to fill in certain values of attributes of a newly created object. In RITA, values of attributes are searched for only if needed to satisfy a subgoal.

4. RITA and MYCIN both have a type of LHS predicate clause which acts like an existential quantifier:

IF: there is a block whose color is blue ...

The RITA system performs a complete backtracking scan in an attempt to satisfy nested existentials. Ultimately, if necessary, all possible relevant data bindings will be attempted to satisfy a set of conditionals. MYCIN does not perform such backtracking under similar conditions.

System Architecture

Our implementation of RITA has been influenced by the factors mentioned in Sec. II: funding source, expected user community, and available computing resources. In this context, we have made the following implementation decisions in designing the software architecture of the RITA system:

1. For efficient operation on the limited resources of a minicomputer, we have decided to "compile" English-like rule and object descriptions into an internal list-structure form for use by the monitor; however, corresponding "decompilers" allow a user, upon request, to see any data in the symbolic, English-like form. Rules and object descriptions are always stored externally (e.g., on disk) in their English-like form.

2. RITA has been designed as two cooperating modules, so that only the currently active module need reside in the core during the operation of a user agent. Such modularity is aided by the facilities in the UNIX operating system for communication between separate processes.

The *user interface* module gives the user one or more windows within which text can be displayed, with the facilities of the Rand Editor available for text manipulation within those windows. It allows creation of rule sets and contexts in a symbolic English-like form, and passes rules and commands to the monitor to allow user control over its operation.

The *monitor* module contains facilities for compiling symbolic form rules and data descriptions into an internal list-structure form and for decompiling internal forms back into symbolic form upon request. It uses one of the available monitors to apply a rule set to a context, and emits trace information to a history file for use by diagnostic and tutorial facilities.

3. We have chosen to use an excellent compiler-compiler, named YACC (Yet Another Compiler-Compiler) and available on the UNIX system, to implement our rule and object compilers. Since all compilation of symbolic forms of rules and objects is governed by a BNF grammar, it is not difficult to make changes in the surface syntax of rule and object descriptions.

A basic form of RITA is currently operational. The following section gives brief descriptions of the features currently implemented in RITA, enhancements planned during the next six to nine months, and research questions raised by our work to date.

V. CURRENT PROGRAM STATUS AND CONCLUDING REMARKS

The following features are implemented in the current (January 1976) version of RITA:

Both pattern-directed and backward-chaining monitor modes operate with unordered rule sets, with the backward-chaining mode capable of being initiated by the pattern-directed mode.

The user can initiate, interrupt, resume, and terminate the monitor's operation. During an interruption, he may add, modify, or delete rules or objects, inspect rules or objects, and set conditions (e.g., the testing of a particular rule's predicates or execution of its actions) upon which monitor operation should be interrupted. He may also obtain information about the recent history of the monitor's operation, such as which rules have been tested and have failed or succeeded, or which object's attribute values have been set.

Attribute values may be primitive values (e.g., character strings) or sets of values. Members of such value sets may themselves be sets, permitting an arbitrarily deep nesting of value data.

The RITA system can interact with external information systems, either via the ARPANET or dial-up access over telephone lines.

RITA agents may be initiated by a reminder facility added by Rand to the UNIX system.* Agents may be "reminded" to start operation at a specific time and date (e.g., at 3 a.m. on March 17, 1976) or at a relative time (e.g., six hours from now, or in three days). These facilities can also be used to awaken agents on a regular periodic schedule (e.g., every day at 3 a.m.). The *remind* function allows agents to periodically check the status of a lengthy remote operation or to initiate routine tasks after normal working hours. Reminders, when created, are written on a special disk file along with such status information as whether they are pending or in operation and the time and

*The *remind* function, which performs very useful functions both for RITA and UNIX system users, was conceived, designed, and implemented by Dr. Steven Zucker of the Rand computer research staff.

date for initiation. Whenever the UNIX system is restarted, for example after a power shut-down for maintenance or a system crash, a system function automatically checks the reminder file for pending actions. In this way, RITA agents are usually capable of operating in pursuit of goals over extended periods of time in spite of interruptions. (An exception is when RITA is interrupted while interacting with external systems. It is often not possible to resume an interaction with an external system at some midpoint in that interaction.)

Principal features discussed in this report but not yet operational within the RITA system are

- o The ability to use levels of confidence on attribute values and
- o The use of the Rand Editor as part of a user's interface to RITA for the creation, modification, and perusal of rules and objects in a user agent.

These capabilities are expected to become available within several months after the publication of this report.

The RITA interpreter module currently occupies a total of about 49,000 bytes of core (28,000 program and 21,000 data). The user interface module is quite small (about 9000 bytes). The two modules can grow independently to the maximum core available. In addition, about 250 bytes are required per rule stored, plus 100 bytes per object stored.* Therefore, our current system, with 64,000 bytes each of program and data storage space available, can grow about 220 percent in program size and can store in-core an agent consisting of (for example) 120 rules operating upon a data base having 130 objects.

User agents comprising about 50 rules are now in use, and can, for example, handle ARPANET file transfer operations. We expect to create agents at least several times this size in the near future.

Many questions are still unanswered at this stage of the research project. For example:

---

*Assuming about 5 clauses per rule and 10 attributes per object.

- o Will a computer-naive user really be able to modify the operation of a user agent by adding or modifying rules? If so, how long a prior familiarization period is required?
- o At what level of size or complexity of a user agent will speed of operation and efficiency become important considerations?
- o What about system security? Knowledge about passwords, access keys, data formats, and account numbers for external systems might well reside within RITA user agents, making the intelligent terminal itself a valuable target for compromise. Even with physical security, such as restricting access to the room containing the terminal, often there will still remain external communication paths to the machine that allow possible access to data. We need to understand more about the constraints which must be placed on access to intelligent terminals containing sensitive data in representative user environments.

In conclusion, we emphasize that this is a report on work in progress, and many questions are left unanswered. However, we are encouraged by our initial experimentation in the use of production systems to represent heuristics governing intelligent terminal behavior. Their ability to provide explanations of that behavior, to be modified and incrementally extended by a user, and to operate in either a pattern-directed or goal-directed manner are all potentially valuable features. We believe the RITA system, whose design we have discussed here, provides a good testbed for the demonstration and evaluation of these intelligent terminal agent capabilities.

Appendix
EXAMPLES OF RITA OPERATION

LHS SCAN OF UNORDERED RULE SET

As an illustration of an unordered rule set to be interpreted by a LHS scan monitor, consider the following set of eight rules. The purpose of these rules is to generate a telephone call to the recipient and to notify the user when the call has been successfully placed. The rules must deal with such exigencies of the telephone system as busy signals and no answer. These rules assume the context contains an object called a "recipient", who has the following attributes:

Object Type	Attribute Name	Example Value	Comment
recipient	name	"Bob Jones"	
	primary_phone#	"(213) 393-0411"	
	alternate_phone#	"(213) 393-0412"	[optional]
	home_phone#	"(714) 454-8812"	[optional]

[optional] means that the attribute need not exist for the recipient.

The rules create an object of type "call" whose attributes record the current knowledge about the attempted call. This object is in turn used by another group of rules which specializes in dealing with the external telephone system through a dialing unit, and which sets a "status" attribute for the call to reflect the information reported by the dialing unit. The "call" object has the following attributes and sample values:

Object Type	Attribute Name	Sample Values
call	status	"to be initiated" "busy" "answered" "not answered" "unacceptable phone#"
	target	"primary phone#" "alternate phone#" "home phone#"
	desired_placement_time	"75/7/20 1800"
	priority	"normal" "urgent"

The rule set is shown below; comments are enclosed in square brackets. The particular syntax used to state the rules is indicative of the form of RITA rules, but is subject to change. The reference manual [Anderson and Gillogly, to be published] should be consulted for the current formal description of allowable RITA rule forms.

```
                    [RULES FOR PLACING A TELEPHONE CALL]

RULE 1          [when to create a new call]

     IF:  there is a recipient whose primary_phone# is known &
          there is not a call

   THEN:  create a call whose status is "to be initiated" &
            whose target is "primary phone#" &
            whose desired_placement_time is currenttime;[†]


RULE 2          [when to actually dial the number]

     IF:  the status of the call is "to be initiated" &
          the desired_placement_time of the call is less than
                  [i.e. earlier than] currenttime

   THEN:  set the status of the call to "ready for dialing";[‡]


RULE 3          [what to do with a busy signal]

     IF:  the status of the call is "busy"

   THEN:  set the desired_placement_time of the call to
                            currenttime + 2 [minutes] &
          set the status of the call to "to be initiated";[††]
```

[†] We assume "currenttime" is a function which returns the current date/time as its value.

[‡] We assume this call status triggers additional rules, not shown here, which interact with a dialing unit to physically dial a call, then read the dialing unit's output signals and set the "status" attribute of the call to one of: "busy", "answered", "not answered", "unacceptable phone#". These other rules are expected to use the "target" attribute of the call to select a phone# from one of the attributes of the recipient.

[††] We use infix notation (e.g., "a+b") in these rules to express arithmetic operations in an easily readable form, although the current version of RITA requires the use of functional notation (e.g., "plus(a,b)").

```
RULE 4          [what to do if the call is answered]

     IF:   the status of the call is "answered"

   THEN:   send concat (the name of the recipient,
                "has been reached at his/her",
                target of the call,
                ".  Please pick up your phone.") to the user &
           delete the call;[†]


RULE 5          [what to do if primary phone# is not answered]

     IF:   the status of the call is "not answered" &
           the target of the call is "primary phone#" &
           the alternate_phone# of the recipient is known

   THEN:   set the target of the call to "alternate phone#" &
           set the status of the call to "to be initiated";


RULE 6          [what to do if alternate phone# is not answered]

     IF:   the status of the call is "not answered" &
           the target of the call is "alternate phone#" &
           the priority of the call is "normal"

   THEN:   set the desired_placement_time of the call to
                       currenttime + 30 [minutes] &
           set the target of the call to "primary phone#" &
           set the status of the call to "to be initiated";


RULE 7          [special rule for an unanswered urgent call]

     IF:   the status of the call is "not answered" &
           the target of the call is "alternate phone#" &
           the priority of the call is "urgent" &
           the home_phone# of the recipient is known

   THEN:   set the target of the call to "home phone#" &
           set the status of the call to "to be initiated";
```

[†]This rule assumes the existence of a "concat" function which evaluates its arguments, then returns a single character string consisting of the concatenation of the argument values.

```
RULE 8          [what to do with an unanswered home call]

    IF:  the status of the call is "not answered" &
         the target of the call is "home phone#"

  THEN:  set the desired_placement_time of the call to
                        currenttime + 15 [minutes] &
         set the status of the call to "to be initiated";
```

The above rules illustrate a fragment of a complete system for interacting with the telephone system. As such, they assume a considerable amount of context and leave several questions unanswered; for example:

These rules assume the person who places the call will be available to handle the call, either now or at some arbitrary time in the future when the call becomes completed. If this is unrealistic, Rule 2 should check for the availability of the caller before actually dialing the call.

The rules for determining the priority of the call are not shown. Presumably, they would encode a heuristic like "assume it's normal unless I tell you explicitly that it's urgent."

These rules might be considerably enhanced by the addition of other rules to determine, given the area code of the target phone number, the time zone of the recipient;[*] it could then be determined whether currenttime in that zone is within scheduled business hours, during the lunch hour, or outside scheduled business hours.

This information could then be used to influence the calling strategy in such situations as an unanswered call.

As an illustration of the relative ease of modifying and extending an existing rule set, suppose that during repeated use of the above agent a user notices a possible flaw in the logic: recipient Steve Kramer does not have an alternate phone number, but he does have a known home phone number. An urgent call is placed to Steve, who does not answer his primary number. No further attempt to place the

[*] We use this logic for simplicity in this illustration. In practice, some area code zones cross time zone boundaries, so more complex logic is needed.

call is made by the agent; in particular, his home number is not tried. By observing the rules used to arrive at this decision, the user realizes that Rule 7 only sets the target of the call to the home number if the alternate number is unanswered. He decides the following two rules will help in this situation:

```
RULE X          [how to handle an unanswered urgent call when there
                   is no alternate number but the home number is
                   known]

    IF:  the status of the call is "not answered" &
         the priority of the call is "urgent" &
         the target of the call is "primary phone#" &
         the alternate_phone# of the recipient is not known &
         the home_phone# of the recipient is known

  THEN:  set the target of the call to "home phone#" &
         set the status of the call to "to be initiated";

RULE Y          [how to handle an unanswered call if no other
                   phone numbers are known]

    IF:  the status of the call is "not answered" &
         the alternate_phone# of the recipient is not known &
         the home_phone# of the recipient is not known

  THEN:  set the desired_ placement time of the call to
                        currenttime + 30 [minutes] &
         set the status of the call to "to be initiated";
```

We note that since this is a rule set in which the order of the rules is not important, the user may include these rules anywhere, e.g., at the end of the existing rule set. We believe that a computer-naive user might well not have easily accomplished the initial eight-rule set to handle his telephone calls, but given that set as an initial starter package--which establishes the vocabulary and basic logic of the approach--we can imagine him adding Rules X and Y by copying the phrases of the original rules with some minor repackaging and modification. This assumption has not yet been tested with actual computer-naive users, but it is part of the RITA design philosophy that this type of incremental enhancement to existing user agents be allowed.

The following is an example of a trace of the above rule-directed user agent in operation. Although various amounts of detail of the operation of this agent might be shown, we show here only the names of rules whose LHS patterns were discovered to be "true," i.e., matched by data in the context, during a hypothesized operation of this agent. We assume that initially there exists in the context a "recipient" with the attributes and values shown at the beginning of this appendix.

Rule Applied	Comment	Interaction With User
Rule 1	creates call object	
Rule 2	invokes rules for sending a call to the phone system	
Rule 5	not answered, so try alternate number	
Rule 2	alternate call tried by invoked rule set	
Rule 6	not answered, so set up to retry primary call 30 minutes later	
Rule 2	after 30-minute delay, primary number tried by invoked rule set	
Rule 3	busy, so set up to retry call 2 minutes later	
Rule 2	after 2-minute delay, primary number tried by invoked rule set	
Rule 4	call is answered	"Bob Jones has been reached at his/her primary phone#. Please pick up your phone."

The user could of course monitor the progress of calls by inserting additional action clauses into those rules that set up a call to be retried later; these action clauses could record statements such as:

"Busy signal on your call to Bob Jones. Will retry in 2 min."
"No answer on your call to Bob Jones. Will retry in 30 min."

If such interaction is too verbose, the user has the option of periodically interrupting the agent and inquiring as to the current values of the status and desired placement time of the call.

GOAL-DIRECTED OPERATION

As an example of goal-directed monitor operation, consider a set of rules which performs the task mentioned in the previous subsection: deciding whether a particular "desired placement time," within the time zone of the recipient, for a call is within or outside scheduled business hours, or during the lunch hour. In describing the logic to be used in making this decision, the following objects, attributes, and values will be used as the vocabulary:

Object	Attribute	Sample Values	Comment
recipient	phone#	"(213) 393-0411"	area code is characters 2-4 of the phone number
	areacode	"213"	extracted from phone number by rules
	time_offset	"15", etc.	time offset of recipient's local time zone from Greenwich Mean Time
caller	areacode	"415"	area code of caller
	time_offset	"13", etc.	time offset of caller's local time zone from Greenwich Mean Time
call	desired_placement_time	"75/7/20 1345"	caller's local time
	time_descriptor	"within bus. hrs" "outside bus. hrs" "lunch hour"	determined by recipient's local time
	recipients_local_time	"75/7/20 1545"	translation of desired placement time into recipient's local time
timezone	name	"Eastern" "Central" "Mountain" "Pacific"	
	time_offset	"12", etc.	from Greenwich Mean Time
	areacode_set	("201", "215",...)	set of area codes in that time zone

NOTE: We assume there are multiple objects of type "time zone" in the context, each having specific data about one time zone of interest.

For simplicity in this example, we have shown only one "phone#" for the recipient, rather than the three possible numbers used in the previous example. In practice, the area code would be selected from the target phone number. We are also ignoring the calendar date of the call, which might be used to distinguish weekdays from weekends, etc.

The type of deductive logic needed to derive the time descriptor for the call from other available information can be diagrammed as follows:

To know the call's time descriptor, we need to know:

- o the recipient's local time (corresponding to the call's desired placement time), and to know that, we need:
    - o the call's desired placement time, which is available, and
    - o the caller's time offset, which is available, and
    - o the recipient's time offset, and to know that we need:
        - o the recipient's time zone, and to know that we need:
            - o the recipient's area code, which is available.

The above type of logic is well suited to a goal-directed approach. The following rules encode the process. Their operation is triggered by the action clause: DEDUCE the time descriptor of the call.

[RULES FOR DETERMINING THE TIME DESCRIPTOR FOR A CALL]

```
Rule A          [when to set the descriptor to "lunch hour"]

    IF:  the recipients_local_time of the call is greater than or
             equal to 1200 &
         the recipients_local_time of the call is less than or
             equal to 1300

  THEN:  set the time_descriptor of the call to "lunch hour";
```

```
Rule B          [when to set the descriptor to "within bus. hrs"]

    IF:   the recipients_local_time of the call is greater than or
              equal to 0800 &
          the recipients_local_time of the call is less than 1200

  THEN:   set the time_descriptor of the call to "within bus. hrs";


Rule C          [another "within bus. hrs" possibility]

    IF:   the recipients_local_time of the call is greater than 1300 &
          the recipients_local_time of the call is less than or
              equal to 1700

  THEN:   set the time_descriptor of the call to "within bus. hrs";


Rule D          [when the descriptor is "outside bus. hrs"]

    IF:   the recipients_local_time of the call is greater than 1700 OR
          the recipients_local_time of the call is less than 0800

  THEN:   set the time_descriptor of the call to "outside bus. hrs";


Rule E          [how to compute the recipient's local time]

    IF:   the time_offset of the recipient is known &
          the time_offset of the caller is known &
          the desired_placement_time of the call is known

  THEN:   set the recipients_local_time of the call to
                    the desired_placement_time of the call +
                    (the time_offset of the caller - the time_offset
                    of the recipient);


Rule F          [how to compute the time offset of the recipient]

    IF:   the areacode of the recipient is a member of the
            areacode_set of a timezone (T)

  THEN:   set the time_offset of the recipient to the time_offset
            of the timezone (T);


Rule G          [how to compute the time offset of the caller]

    IF:   the areacode of the caller is a member of the
            areacode_set of a timezone (T)

  THEN:   set the time_offset of the caller to the time_offset of
            the timezone (T);
```

```
Rule H          [how to find the area code of the recipient]

     IF:  the phone# of the recipient is known

   THEN:  set the areacode of the recipient to substr(2,3, phone# of
            the recipient);†
```

To a programmer, the above rule set may seem an excessive amount of verbiage to encode rather simple logic. However, the rule set possesses (at least) two important features: it can be modified and extended quite easily, and a monitor can use it to give an explanation of its behavior.

This rule set could be extended, as mentioned above, by adding rules for determining whether a date is a weekday or a weekend. The given rules for setting the time descriptor to either "within bus. hrs" or "lunch hour" would then be given an additional clause requiring the condition that the date of the call be a weekday.

As an example of possible trace information which could be generated from the above rule set in response to a user's requests, consider the following excerpt of dialog. (RITA's responses are shown capitalized only to distinguish them from the user's comments.)

```
  User              RITA

. . .

What did you do?

   THE TIME_DESCRIPTOR OF THE CALL WAS SET TO "outside bus. hrs"

Why?

   BECAUSE RULE D WAS APPLIED:

   IF:  THE RECIPIENTS_LOCAL TIME_ OF THE CALL IS GREATER THAN 1700 OR
        THE RECIPIENTS_LOCAL_TIME OF THE CALL IS LESS THAN 0800

   THEN:  THE TIME_DESCRIPTOR OF THE CALL IS "outside bus. hrs";
```

† We assume the availability of a "substring" function; substr(i,n,x) extracts n characters from string x starting at location i.

```
What was the recipients_local_time of the call?

   1745

Why?          [interpreted as:  how did it get set to that time?]

   BECAUSE RULE E WAS APPLIED:

   IF:  THE TIME_OFFSET OF THE RECIPIENT IS KNOWN &
        THE TIME_OFFSET OF THE CALLER IS KNOWN &
        THE DESIRED_PLACEMENT_TIME OF THE CALL IS KNOWN

   THEN:  THE RECIPIENTS_LOCAL_TIME OF THE CALL IS
                 THE DESIRED_PLACEMENT_TIME OF THE CALL +
                 (THE TIME_OFFSET OF THE CALLER - THE TIME_OFFSET
                 OF THE RECIPIENT);

What was the desired_placement_time of the call?

   ...etc.
```

We note that in addition to building a user's confidence in, and familiarity with, the operation of his system, the "why?" facility also helps him create and debug new or modified rule sets. Although at a high level, he is still programming and probably must debug in some form whenever a significant change is made to a rule set.

REFERENCES

Anderson, R. H., and J. J. Gillogly, *Rand Intelligent Terminal Agent (RITA): Reference Manual*, The Rand Corporation, R-1808-ARPA (to be published).

Davis, R., and J. King, *An Overview of Production Systems*, Computer Sciences Department, Stanford University, STAN-CS-75-524, October 1975.

Griswold, R. E., and M. T. Griswold, *A SNOBOL-4 Primer*, Prentice-Hall, Inc., Eaglewood Cliffs, N. J., 1973.

Hopcroft, J. E., and J. D. Ullman, *Formal Languages and Their Relation to Automata*, Addison-Wesley Publishing Co., Inc., Reading, Mass., 1969.

Miller, L. A., "Naive Programmer Problems With Specification of Transfer-of-Control," *Proceedings of the National Computer Conference*, Vol. 44, AFIPS Press, Montvale, N. J., 1975, pp. 657-663.

Moran, T. P., *The Symbolic Imagery Hypothesis: A Production System Model*, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pa., December 1973.

Ritchie, D. M., and K. Thompson, "The UNIX Time-Sharing System," *Communications of the ACM*, Vol. 17, No. 7, July 1974, pp. 365-375.

Shortliffe, E. H., S. G. Axline, B. G. Buchanan, T. C. Merigan, and S. N. Cohen, "An Artifical Intelligence Program to Advise Physicians Regarding Antimicrobial Therapy," *Computers and Biomedical Research*, Vol. 6, 1973, pp. 544-560.

Shortliffe, E. H., S. G. Axline, B. G. Buchanan, and S. N. Cohen, "Design Considerations for a Program to Provide Consultations in Clinical Therapeutics," *Proc. 13th San Diego Biomedical Symposium*, February 4-6, 1974, pp. 311-319.

Shortliffe, E. H., *MYCIN: A Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection*, Computer Science Department, Stanford University, Stanford, Ca., STAN-CS-74-465, October 1974. (Also available as Stanford Artificial Intelligence Laboratory Memo AIM-251. A condensed version will be published as *MYCIN: Computer-based Medical Consultations*, American Elsevier Publishing Co., Inc., New York, 1976.)

Shortliffe, E. H., and B. G. Buchanan, "A Model of Inexact Reasoning in Medicine," *Mathematical Biosciences*, Vol. 23, 1975, pp. 351-379.

Shortliffe, E. H., R. Davis, S. G. Axline, B. G. Buchanan, C. C. Green, and S. N. Cohen, "Computer-Based Consultations in Clinical Therapeutics: Explanation and Rule-Acquisition Capabilities of the MYCIN System," *Computers and Biomedical Research*, Vol. 8, 1975, pp. 303-320.

Swinehart, D. C., *COPILOT--A Multiple Process Approach to Interactive Programming Systems*, Computer Science Department, Stanford University, Stanford, Ca., STAN-CS-74-412, July 1974.

Waterman, D. A., "Generalization Learning Techniques for Automating the Learning of Heuristics," *Artificial Intelligence*, Vol. 1, No. 1 and 2, 1970, pp. 121-170.

Waterman, D. A., *PAS-II Reference Manual*, Computer Science Department Report, Carnegie-Mellon University, Pittsburgh, Pa., June 1973.

Waterman, D. A., *Adaptive Production Systems*, Psychology Department, Carnegie-Mellon University, Pittsburgh, Pa., CIP Working Paper 285, December 1974.